2010 39th International Conference on Parallel Processing Workshops

Improving the Effectiveness of Context-based Prefetching with Multi-order Analysis [♯]

Yong Chen [§]
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, Tennessee USA
Email: cheny@ornl.gov

Huaiyu Zhu, Hui Jin, and Xian-He Sun
Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois USA
Email: {hzhu12, hjin6, sun}@iit.edu

Abstract

Data prefetching is an effective way to accelerate data access in high-end computing systems and to bridge the increasing performance gap between processor and memory. In recent years, the context-based data prefetching has received intensive attention because of its general applicability. In this study, we provide a preliminary analysis of the impact of orders on the effectiveness of the context-based prefetching. Motivated by the observations from the analytical results, we propose a new context-based prefetching method named Multi-Order Context-based (MOC) prefetching to adopt multi-order context analysis to increase the context-based prefetching effectiveness. We have carried out simulation testing with the SPEC-CPU2006 benchmarks via an enhanced CMP$im simulator. The simulation results show that the proposed MOC prefetching method outperforms the existing single-order prefetching and reduces the data-access latency effectively.

Keywords: data prefetching, context-based prefetching, memory access performance, high-end computing.

[♯] *This research is sponsored in part by the National Science Foundation under NSF grant CCF-0621435 and CCF-0937877, by the ACM/IEEE High-Performance Computing Ph.D. Fellowship. This research is also sponsored in part by the Office of Advanced Scientific Computing Research; U.S. Department of Energy. The work was performed in part at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.*

[§] *This work was primarily performed while he was at Illinois Institute of Technology.*

1. Introduction

The rapid advance of semiconductor process technology allows the processor speed or the aggregate processor speed on chip with multicore/manycore architectures grows fast and steadily. The memory speed or the data load/store performance, on the other hand, has been increasing at a snail's pace [11]. The memory speed has only increased by roughly 9% each year over the past two decades, which is significantly lower than the improvement speed of nearly 50% per year for processor performance [11]. This trend is predicted to continue in the next decade. This unbalanced performance improvement leads to one of the significant performance bottlenecks in high-end computing known as "memory-wall" problem [18][28]. Multiple memory hierarchies have been the primary solution to bridging the processor-memory performance gap. However, due to the limited cache capacity and highly associative structure, large amount of off-chip accesses and long data-access latency still spike the performance severely.

Data prefetching approach was thus proposed to reduce the processor stall time when applications lack temporal or spatial locality, and has been widely recognized as a critical companion technique of memory hierarchy solution to overcome the memory-wall issue [18][28][11]. As the term indicates, the essential idea of data prefetching is to observe data referencing patterns, then speculate future references and fetch the predicted reference data closer to processor before processor demands them. Numerous studies have been conducted and a lot of strategies have been proposed for data prefetching [1][2][5][13][14][15][19][20][27]. These studies concluded that data prefetching is a promising solution to reducing memory access latency.

1530-2016/10 $26.00 © 2010 IEEE
DOI 10.1109/ICPPW.2010.64

In addition, many commercial high-performance processors have adopted data prefetching techniques to hide long data-access latency [9][11][16].

Among many prefetching strategies, the *context-based data prefetching* has received attention in recent years due to its general applicability and high accuracy [10][22][23]. Although numerous studies have been conducted in context-based prediction and prefetching [10][22][24][25][23], many issues remain open. In this research, we provide a preliminary study on analyzing the impact of *orders*, the length of the context considered, on the prefetching effectiveness via simulation testing. Based on the analytical results, we identify one limitation of the existing context-based prefetching is that they only support a single-order context analysis and prediction. Even though such an approach can achieve high prefetching accuracy, our study shows that the single-order approach leads to limited prefetching coverage. The ultimate goal of a data prefetching strategy is to reduce access delay; however, the performance gain of data prefetching depends on both prefetching coverage and accuracy. The limited prefetching coverage of existing context-based prefetching methods, in turn, leads to limited prefetching effectiveness. While computing capability still increases with a much faster pace than memory performance, more aggressive prefetching strategies are desired, which provide wider coverage and higher accuracy. Motivated by the observations, we propose a new context-based prefetching method named *Multi-Order Context-based (MOC) prefetching* to address the drawback of existing context-based prefetching and to increase the overall prefetching effectiveness.

The rest of this paper is organized as follows. Section II introduces the basic idea of context-based prefetching and reviews important related works. The preliminary analysis of context-based prefetching performance is also presented in Section II. Section III introduces the design of the proposed MOC prefetching strategy and the prefetching methodology. Section IV discusses our simulation experiments and performance results. Section V summarizes this study and discusses the future work.

2. Context-Based Data Prefetching and Preliminary Analysis

In this section, we briefly introduce the context-based data prefetching and review related works. We also present the preliminary analysis of the performance of context-based prefetching that motivates the proposed MOC prefetching strategy.

2.1. Context-Based Data Prefetching and Related Works

The essential idea of the context-based data prefetching is to detect the correlation between current context (the miss access information) and the past history and make predictions for data prefetching. A context-based prefetching method usually builds a state transition diagram with the access address strides (deltas) as states, and characterizes the correlation among miss address streams. Depending on the length of the context considered, the prefetching strategy can adjust the prefetching overhead and confidence in prefetching. The length of context is referred as the *order* of a context-based prefetching strategy.

The context-based prefetching is considered by many as a more generalized approach compared to conventional and classic prefetching strategies including sequential, stream, stride, Markov and distance prefetching. Sequential prefetching strategy brings one or more blocks that follow the current missing block [5][6]. This prefetching mechanism takes advantage of spatial locality and assumes the applications usually request consecutive memory blocks. Stream prefetching can track multiple different streams and prefetch data blocks for each stream [13][16][21]. Stride prefetching [2] detects the stride patterns in data access streams. Once a stride pattern is found, blocks can be prefetched according to the stride detected. Due to its simplicity and effectiveness, stride prefetching is widely used [2][9]. Markov prefetching was proposed in [14] to capture the correlation between cache misses and prefetch data based on a state transition diagram. Distance prefetching [15] combines Markov prefetching and stride prefetching and uses miss strides, not the miss addresses themselves, to build the state transition diagram for detecting the correlation. More recently, the Global History Buffer (GHB) based prefetching [20], Data Access History Cache (DAHC) based prefetching [3], stream chaining prefetching [8], feedback directed prefetching [27] and algorithm-level adaptive prefetching [4] were proposed to further explore the correlation among access history and to provide intelligent, comprehensive and adaptive prefetching strategies.

The generalized context-based prefetching is similar to the context-based value predictor, but is used as a data prefetcher on the cache level. The Finite Context Method (FCM) [24] is a representative context-based predictor that predicts the next value based on a finite number (order) of preceding values. The FCM is usually implemented with a two-level table organization [25], with Value History Table (VHT) and

Value Prediction Table (VPT). The VHT contains the context of data accesses, and the VPT contains the predictions associated with context. A hash function uses the context information from VHT and processor state to form an index that leads to a VPT entry.

A variation of the FCM prefetcher, called Differential Finite Context Method (DFCM), was proposed in [10]. In this model, the context is formed by the differences between values instead of values themselves. DFCM can find more repeating patterns than FCM does, and in the meantime, it reduces the number of VPT entries needed. P-DFCM [22] is a recently proposed data prefetcher based on DFCM. There are two major differences between P-DFCM and DFCM. First, P-DFCM prefetches on L2 load misses only, while DFCM prefetches on both load and store misses. Second, P-DFCM can prefetch when the PC (Program Counter) is known, while DFCM prefetches when both PC and its requesting data address are known. The distance prefetching, originally proposed for TLB [15] and also was used in data cache [19][20], is usually considered as the most representative context-based prefetching where the order is one.

2.2. Preliminary Analysis of Context-Based Prefetching Performance

In this subsection, we present the analysis of context-based prefetching performance via simulation testing. The goal of the performance analysis is to investigate the impact of orders, a critical factor that affects the context-based prefetching performance. We analyze the impact of orders and evaluate the performance from two aspects: the *accuracy*, which measures how many percentages of prefetches are correct, and the *coverage*, which measures how many patterns are recognized and how many misses are reduced. We adopt an architecture-independent approach by collecting miss stream samples of all 29 SPEC-CPU2006 benchmarks [26] and replaying them to analyze the performance of context-based prefetching with different orders.

The prefetching accuracy result of the context-based prefetching for 10 selected representative CPU2006 benchmarks with different orders is shown in Fig. 1, and the average result of all 29 benchmarks is shown in Fig. 2. As can be observed from the results, for the majority of the benchmarks (23 out of total 29), the accuracy increases strictly with the increase of the order, which means that a greater order can achieve higher prediction accuracy. It makes sense since the longer the repeating patterns are, the stronger prediction confidence can be guaranteed.

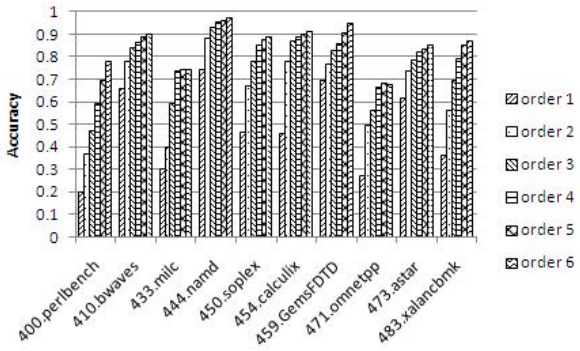

Figure 1. Prefetching Accuracy with Different Orders for 10 Selected Representative SPEC-CPU2006 Benchmarks

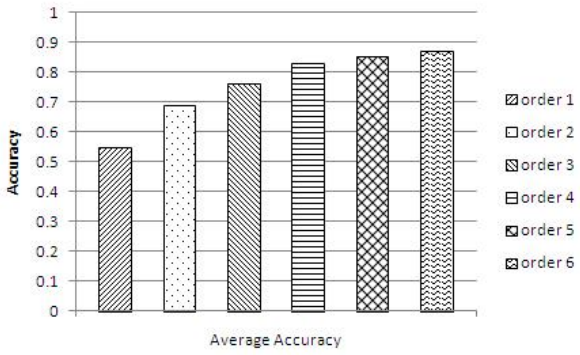

Figure 2. Average Prefetching Accuracy with Different Orders for SPEC-CPU2006 Benchmarks

The prefetching accuracy is not the only metric we care about because there may not be enough effective prefetches to reduce the misses and to hide the latency. We adopt another metric, prefetching coverage, to measure the number of patterns recognized by a prefetcher to evaluate the percentage of misses reduced among all misses. The prefetching coverage of the context-based prefetching with different orders for 10 selected representative CPU2006 benchmarks are shown in Fig. 3, and the average result of all 29 benchmarks is shown in Fig. 4. As can be observed from the results, the lower order model achieves considerably higher coverage. This observation means that a low order model can reduce more misses on the same given sequence of misses than high order models. This is an important observation. This observation means that, even though a high order model can achieve high accuracy, it may not reduce considerable amount of misses. This shows that, even though high accuracy is achieved, the miss reduction could be low, as a

prefetcher could issue prefetches only when it is highly confident.

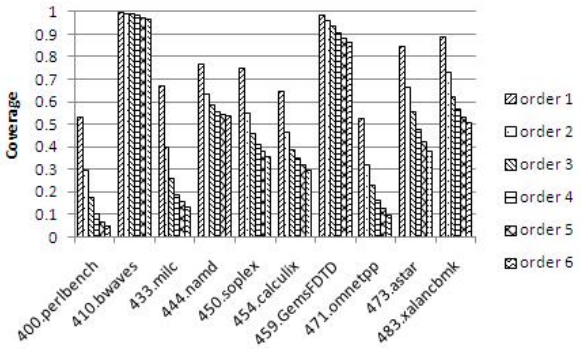

Figure 3. Prefetching Coverage with Different Orders for 10 Selected Representative SPEC-CPU2006 Benchmarks

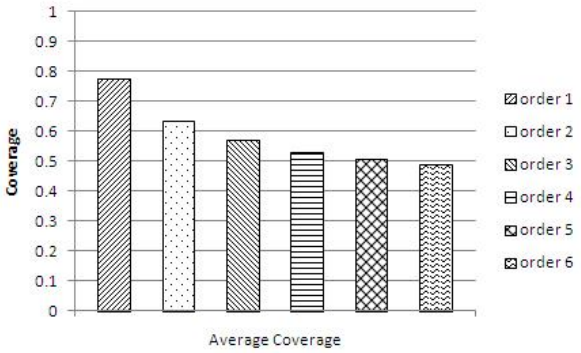

Figure 4. Average Prefetching Coverage with Different Orders for SPEC-CPU2006 Benchmarks

These performance trends for various orders, as shown in Fig. 1 to Fig. 4, demonstrate the need to support multiple orders in order to have the merits of both high accuracy and wide coverage. In the following section, we present the design and prefetching methodology of the proposed MOC prefetching that supports multi-order analysis by taking advantage of a recently proposed prefetching dedicated cache structure, Data-Access History Cache (DAHC) [3].

3. Multi-Order Context-based Prefetching

In this section, we introduce the Multi-Order Context-based prefetching design and its prefetching methodology.

3.1. Multi-Order Context-based Prefetching Design Rationale

We adopt the generic and prefetching-dedicated cache structure, Data-Access History Cache [3], and extend it with context-based prefetching logic to design a multi-order context-based prefetcher.

The proposed multi-order context-based (MOC) prefetching has a three-level table organization as shown in Fig. 5. The first two levels follow the design of the Data-Access History Cache [3], which includes a Data Access History (DAH) table and index tables, PC Index Table (PIT) and Address Index Table (AIT). The DAH table stores the detailed data access history information. It gives high priority to recent access history, and thus filters outdated history automatically. The index tables provide the view and the detection of the correlations from the instruction stream and address stream respectively.

The third level table is the Data Access Prediction (DAP) table. Each DAP entry includes two fields. The first one stores the predicted address corresponding to the context indicated by the entry index. The second field stores a confidence counter that is used to represent how strong the prediction is. DAP entries are indexed by the hashed form of the contexts derived from a hash function. In the current design, we select FS-5 hash function to produce the hashed context as it has better performance than other choices [25].

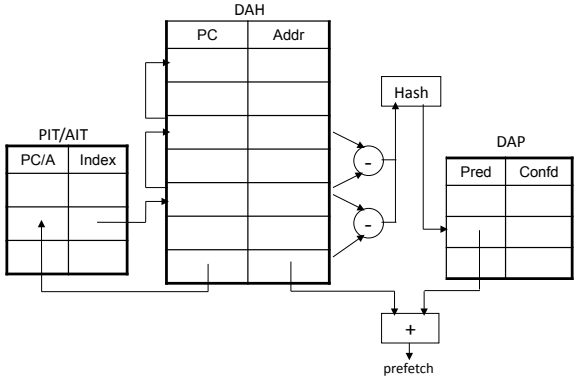

Figure 5. Multi-Order Context-based Prefetching Strategy

3.2. Selection of Multiple Orders

The major constraint for the selection of a specific order comes from the hardware storage requirement for supporting the context analysis with a specific order. The storage requirement, in turn, can be characterized

with the table entry consumption for storing the context and prediction addresses in an ideal case without any hash conflict. In order to have a better understanding of the choice of multiple orders, we analyze the table entry consumption for different orders in an ideal case, as shown in Fig. 6.

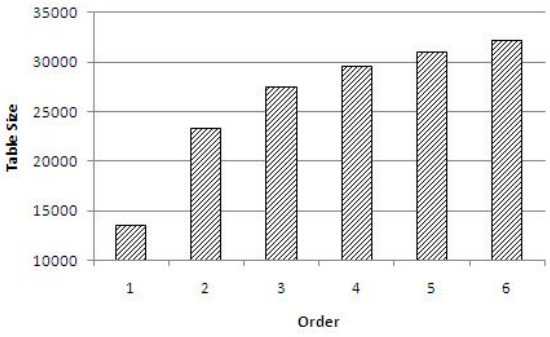

Figure 6. Table Entry Consumption for Different Orders in An Ideal Case

This analysis shows that the higher order we choose, the more table entries we need. Since we use hash table (with FS-5 hash function) in the context analysis, a higher order context analysis can cause more aliasing if the storage budget remains fixed. With the consideration of storage consumption analysis result and the performance of different orders shown in Fig. 1 to Fig. 4, we adopt order-0, order-1 and order-2, not other higher orders, in our current MOC prefetching design. Another important reason for us to choose these three orders in the MOC prefetcher is that we can support order-0 and order-1 prefetching directly by taking advantage of the DAH table without involving additional DAP tables. The order-0 context prefetching means that the prefetcher works with zero number of context (without context). This prefetching method is essentially a sequential prefetching, which is optimal for programs that access consecutive memory blocks. The order-1 context prefetching means that the prefetcher makes the prediction based on one-level access stride, which is essentially the existing distance prefetcher [15][19]. In theory, multiple DAP tables are needed so that different order contexts can be detected by the prefetcher, which requires higher hardware investment. Fortunately, a great advantage of MOC prefetching is that it can carry out low-order context prefetching including order-0 and order-1 prefetching without additional DAP tables. This is because that the DAH table maintains the full information of the recent data accesses. We can obtain the order-0 analysis (past accesses) and order-1 analysis (strides among accesses) directly from the DAH table.

3.3. Multi-Order Context-based Prefetching Mechanism

The mechanism of multiple order context-based prefetching is described as follows. Upon a cache miss, the order-0 prefetching issues *n* blocks of data requests, depending on the prefetching degree, following the miss address. The order-1 prefetching is similar to the distance prefetching based on DAHC. The order-2 prefetching uses the sequence of strides between miss addresses as the context as shown in Fig. 5. When a new data access is captured in DAH, the prefetcher searches the PIT to find the last miss address from the same PC. It then follows the PC chain to retrieve the miss sequence from the same PC. The recent two strides are fed into the hash function unit and the output is a hashed form of context. The hashed context in our current design consists of 9 bits. It is used as an index for looking up DAP tables. If an entry can be found in DAP tables with a predicted stride in its prediction field, prefetch requests are generated and issued. The confidence counter associated with each DAP entry reflects the confidence of the prediction following a certain context. The confidence counter (2 bits) has four values (0, 1, 2 and 3) indicating lowest, low, high and highest confidences. The counter value increases upon a correct prediction and decreases upon a false prediction. The equivalent state diagram and state transition for the confidence counter is shown in Fig. 7. When the confidence counter indicates a high or highest confidence, i.e. value 2 or 3, a prefetch is generated and the prefetch address is calculated as current miss address plus predicted stride.

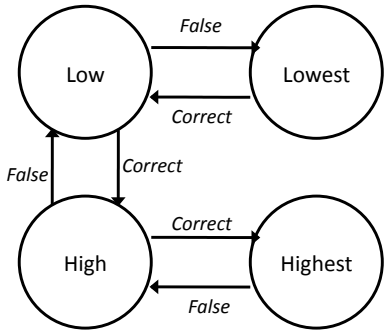

Figure 7. State Diagram for Confidence Counter

4. Simulation and Performance Results

We have carried out the simulation of the proposed multi-order context based prefetching to evaluate its

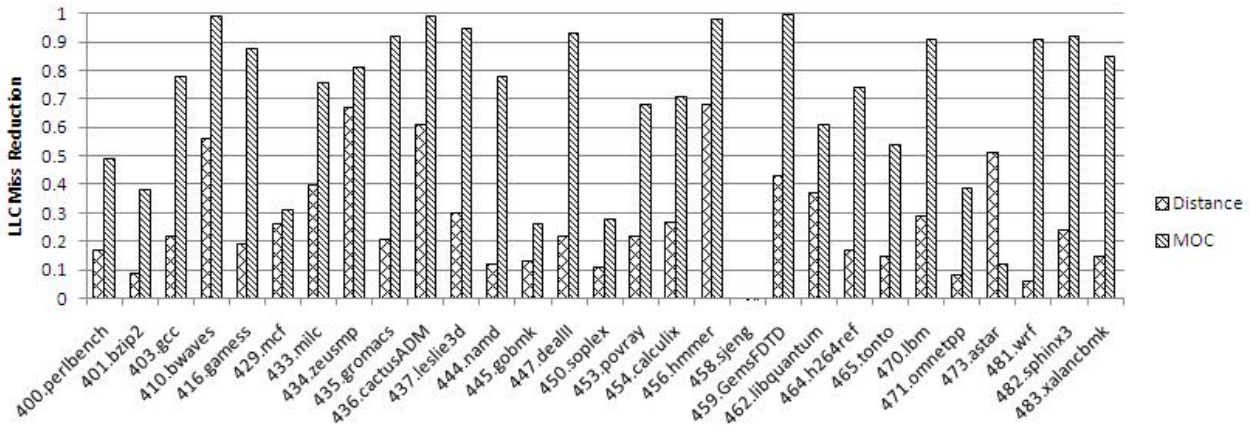

Figure 8. L2 Cache Miss Rate Reduction of the Distance Prefetching and MOC Prefetching

performance by using Pin trace collection tool [17] and CMP$im simulator[12]. In this section, we present the simulation results and compare its performance with the existing single order-1 distance prefetching [19][15].

4.1. Simulation Configuration

Pin [17] is a dynamic instrumentation tool that can collect program traces, and CMP$im [12] is a trace-driven simulator that characterizes the memory system performance for both single-threaded and multi-threaded workloads. The first Data Prefetching Competition (DPC-1) committee [7] released a prefetcher kit that provides partial interface to make it feasible to integrate CMP$im simulator with an add-on prefetching module. We utilize this prefetcher kit to simulate MOC prefetching strategy and evaluate its performance.

We have conducted the simulation testing with all 29 SPEC-CPU2006 benchmarks [25]. The benchmarks were compiled using GCC 4.1.2 with -O2 optimization. We collect traces for all benchmarks by fast forwarding 40 billion instructions and then running 100 million instructions, which is widely used in evaluating an architectural enhancement. We use the *ref* input size for all benchmarks. The simulator was configured as an out-of-order issue processor with a 15-stage, 4-wide pipeline (maximum of two loads and maximum of one store can be issued every cycle).The detailed simulation settings are listed in Table 1. The MOC prefetching is configured as follows. The DAH table, the index table and the DAP table have 1024, 512 and 512 entries respectively. The single order-1 distance prefetcher we simulate for performance comparison uses a 1024-entry distance prediction table [19][15].

Table 1. Simulation Configuration

Parameter	Value
Window Size	128-entry
Issue Width	4
L1 Cache	32KB, 8-way
L2 Cache	512KB/1MB/2MB, 16-way
Block Size	64B
L2 Cache Latency	20 cycles
Memory Latency	200 cycles
L2 Cache Bandwidth	1 cycle/access
Memory Bandwidth	10 cycles/access

4.2. Performance Analysis

Fig. 8 shows the percentage of L2 cache misses reduced by the MOC prefetching and the distance prefetching, respectively. On average, MOC reduces L2 cache misses by over 65%, which is more than two times of what the distance prefetching can achieve. Cache misses are reduced significantly for most benchmarks including five benchmarks whose misses are reduced over 90%. 458.sjeng is the only benchmark on which MOC prefetching has had negative performance.

Fig. 9 reports the IPC (Instructions per Cycle) speedup results. The bars shown in each group include the base case (without prefetching), distance prefetching and MOC prefetching, from left to right. The simulation result shows that the MOC prefetching significantly reduces the average data access latency and improves IPC considerably. The IPC performance improvement is peaked at 285%, and the overall average IPC improvement of all 29 benchmarks is nearly 28%. The MOC prefetching outperforms the distance prefetching considerably, which achieves an average 9.1% IPC speedup.

It can be observed that the MOC prefetching perfor-

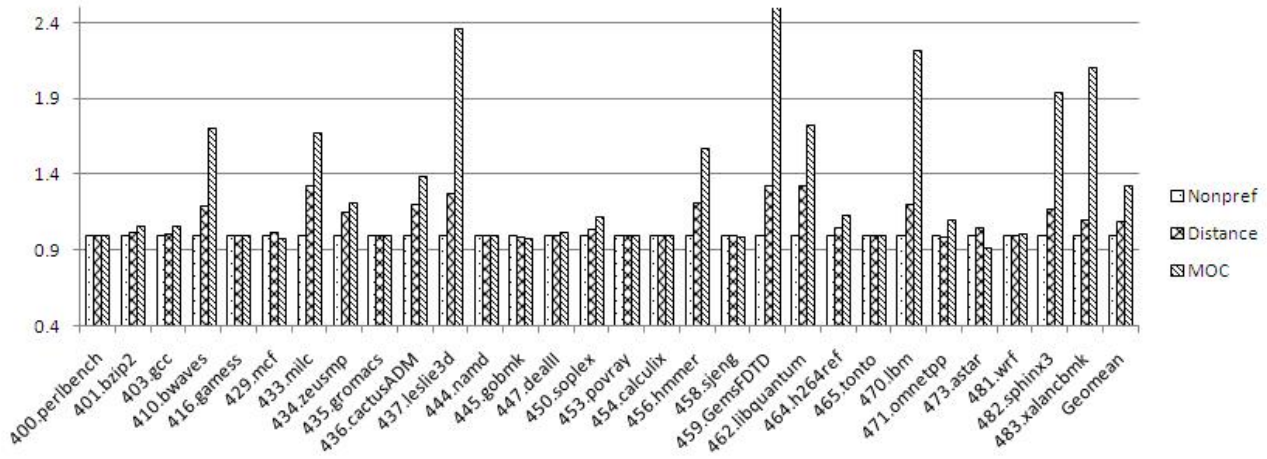

Figure 9. IPC Speedup of the Distance Prefetching and MOC Prefetching

mance is related with the miss rate of the benchmarks and it achieves better performance when dealing with benchmarks with high miss rate. There are a few exceptions including 403.gcc (with miss rate 75%) and 429.mcf (with miss rate 97%). The detailed analysis shows that the reason for the 403.gcc benchmark is that it has only 0.2M accesses over 100M instructions. Applications with infrequent data access, such as 403.gcc, can hardly be improved by prefetchers even their miss rate is high. 429.mcf is another extreme case. It has a considerably large amount of L2 cache accesses (5.1M) and misses (5M) during 100M instructions compared with other benchmarks. Highly frequent data accesses compound with high miss rate reduces the efficiency of prefetchers too. In addition to 429.mcf, another three benchmarks, 445.gobmk, 458.sjeng and 473.astar, also experience a slight performance slow down. Since their performance differences are no more than 8%, we consider them acceptable.

5. Conclusion and Future Work

As memory access performance lags far behind computational performance, data access delay has a severe impact on overall system performance. The recent advance in multicore and manycore processor architectures has put more pressure than ever on reducing data access delay for high-end computing. Data prefetching mechanisms, especially the general context-based prefetching approach, have received intensive attention and have been proven effective in masking the processor-memory performance gap. This study targets to further improve the effectiveness of context-based prefetching by exploiting multi-order context analysis.

The contribution of this research is three-fold. First, we have conducted an analytical study on comparing different orders of context-based prefetching models. Second, we propose a new multi-order context-based (MOC) data prefetching that incorporates multi-order context analysis to achieve better overall prefetching effectiveness. Third, we have conducted simulation testing to validate the MOC prefetching design and to evaluate the performance improvement.The simulation with SPEC-CPU2006 benchmarks demonstrates the advantage of multiple-order analysis in the context-based prefetching. They show that the proposed MOC prefetching outperforms the existing single order-1 context-based prefetcher, distance prefetcher. The MOC prefetching achieves considerable latency reduction and is promising in hiding data access latency for high-end computing systems.

In the near future, we plan to investigate the selection of multiple orders more intelligently and dynamically. In addition, we will continue exploring the combination of multi-order context analysis and global context analysis - the correlation from all instructions - to further improve context-based data prefetching.

References

[1] S. Byna, Y. Chen and X.-H. Sun. A Taxonomy of Data Prefetching Mechanisms. Intl. Symp. on Parallel Architectures, Algorithms, and Networks, 2008.

[2] T.F. Chen and J.L. Baer. Effective Hardware-Based Data Prefetching for High Performance Processors. IEEE Trans. Computers, pages 609-623, 1995.

[3] Y. Chen, S. Byna, and X.-H. Sun. Data Access History Cache and Associated Data Prefetching mechanisms. ACM/IEEE SuperComputing Conference, 2007.

[4] Y. Chen, H. Zhu and X.-H. Sun. An Adaptive Data Prefetcher for High-Performance Processors. Accepted to appear in the Proc. of the 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2010.

[5] F. Dahlgren, M. Dubois, and P. Stenstrom. Fixed and Adaptive Sequential Prefetching in Shared-memory Multiprocessors. International Conference on Parallel Processing, pages I56-I63, 1993.

[6] F. Dahlgren, M. Dubois, and P. Stenstrom. Sequential Hardware Prefetching in Shared-Memory Multiprocessors. IEEE Trans. on Parallel and Distributed Systems, pages733-746, 1995.

[7] DPC-1 Homepage, 2008. http://www.jilp.org/dpc/

[8] P. Diaz and M. Cintra. Stream Chaining: Exploiting Multiple Levels of Correlation in Data Prefetching. Intl. Symp.on Computer Architecture, 2009.

[9] J. Doweck. Inside Intel Core Micro-architecture and Smart Memory Access. Intel White Paper, 2006.

[10] B. Goeman et al. Differential FCM: Increasing Value Prediction Accuracy by Improving Table Usage Efficiency. Intl. Symp. on High performance Computer Architecture, pages 207-218, 2001.

[11] J. Hennessy and D. Patterson. Computer Architecture: A Quantitative Approach. The 4th edition, Morgan Kaufmann, 2006.

[12] Aamer Jaleel, Robert S. Cohn, Chi-Keung Luk, and Bruce Jacob. CMP$im: A Pin-based On-the-fly Multi-core Cache Simulator. Fourth Annual Workshop on Modeling, Benchmarking and Simulation, 2008.

[13] N. P. Jouppi. Improving Direct-mapped Cache Performance by the Addition of a Small Fully-associative Cache and Prefetch Buffers. Intl. Symp. on Computer Architecture, 1990.

[14] D. Joseph and D. Grunwald. Prefetching Using Markov Predictors. Intl. Symp. on Computer Architecture, 1997.

[15] G. B. Kandiraju and A. Sivasubramaniam. Going the Distance for TLB Prefetching: An Application-driven Study. Intl. Symp. on Computer Architecture, 2002.

[16] H. Le, W. Starke, J. Fields, F. O'Connell, D. Nguyen , B. Ronchetti, W. Sauer, E. Schwarz, M. Vaden. The POWER6 Microarchitecture, IBM Technical White Paper, 2007.

[17] C. K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. Programming Language Design and Implementation, 2005.

[18] S. A. McKee. Reflections on the Memory Wall. Conf. Computing Frontiers, 2004, p. 162.

[19] K. J. Nesbit, A. Dhodapkar, and J. E. Smith. AC/DC: An Adaptive Data Cache Prefetcher. Paralell Architectures Compilation Techniques, 2004.

[20] K. J. Nesbit and J. E. Smith. Prefetching Using a Global History Buffer. Intl. Symp. on High Performance Computer Architecture, pages 96-106, 2004.

[21] S. Palacharla and R. E. Kessler. Evaluating stream buffers as a secondary cache replacement. Intl. Symp. on Computer Architecture, 1994.

[22] L. Ramos, J. Briz, P. Ibanez, V. Vinals. Data Prefetching in a Cache Hierarchy with High Bandwidth and Capacity. ACM Computer Architecture News, pages 37-44, 2007.

[23] L. Ramos, J. Briz, P. Ibanez, V. Vinals. Multi-level Adaptive Prefetching based on Performance Gradient Tracking. The 1st JILP Data Prefetching Championship, 2009.

[24] Y. Sazeides and J. E. Smith. The predictability of data values. Intl. Symp. on Microarchitecture, 1997.

[25] Y. Sazeides and J. E. Smith. Implementations of context based value predictors. TR ECE97-8, Dept. of Electrical and Computer Engineering, Univ. Wiscosin-Madison, 1997.

[26] Cloyce D. Spradling. SPEC CPU2006 Benchmark Tools. SIGARCH Comput. Archit. News, 2007.

[27] S. Srinath, O. Multu, H. Kim, Y. N. Patt. Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers. Intl. Symp. on High Performance Computer Architecture, 2007.

[28] W. A. Wulf and S. A. McKee. Hitting the Memory Wall: Implications of the Obvious. ACM SIGARCH Computer Architecture News, 23(1):20-24, 1995.